softserve experience matters

Java Core

Multithreading IO Streams

IT Academy
05/2016

Agenda

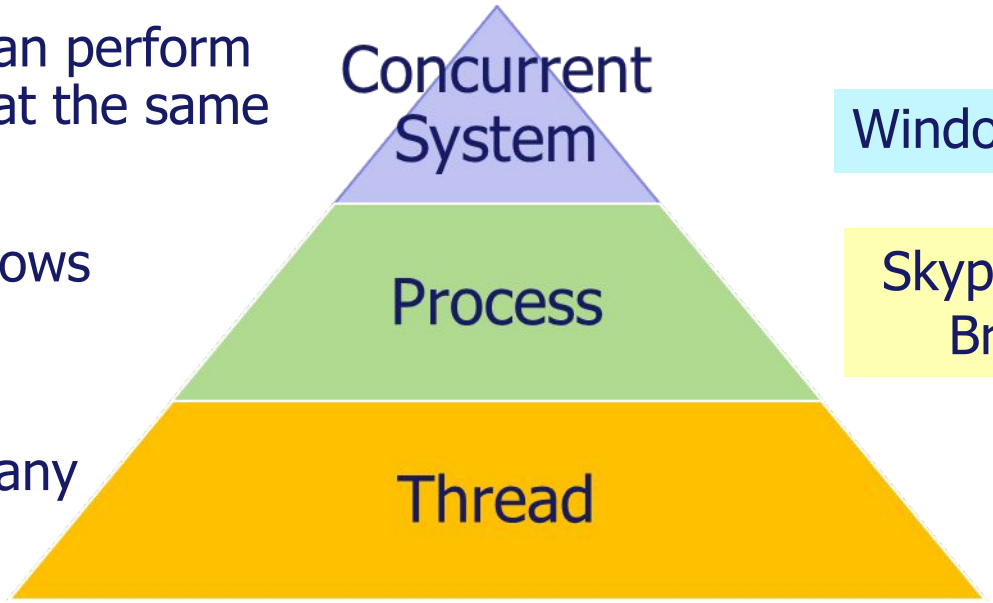
- Processes and Threads
- Threads in Java
- Java Input and Output Streams
- File Input/Output streams
- Practical tasks

What exactly is a concurrent ?

A system is concurrent if it can perform several activities in parallel (at the same time)

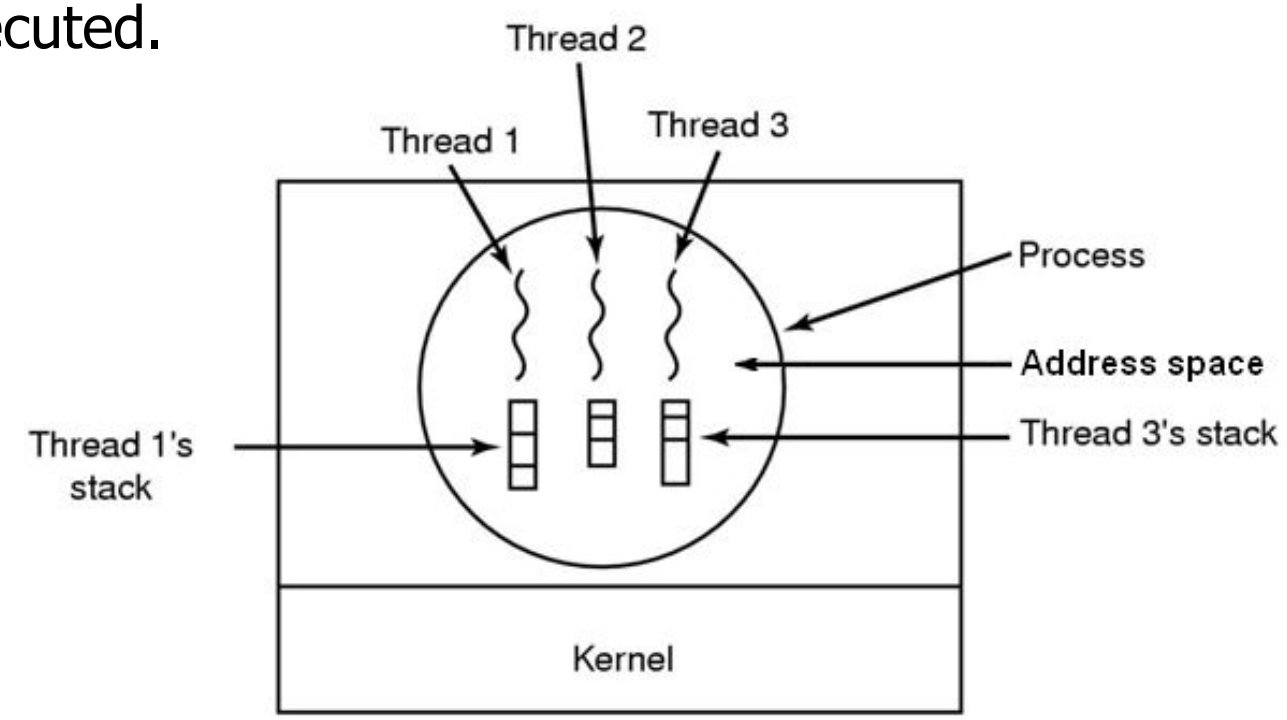
Modern OS like Unix or Windows support multiple processes ("multitasking")

- A process can have many threads
- share data
- Influence each other
- Java threads are managed by JVM
- Each program starts with at list one thread ("main")



Processes and Threads

Process is a set of threads within process' address space
Each thread has its own set of CPU registers, called the thread's context.
The context reflects the state of the thread's CPU registers when the thread last executed.



How to create new Thread ?

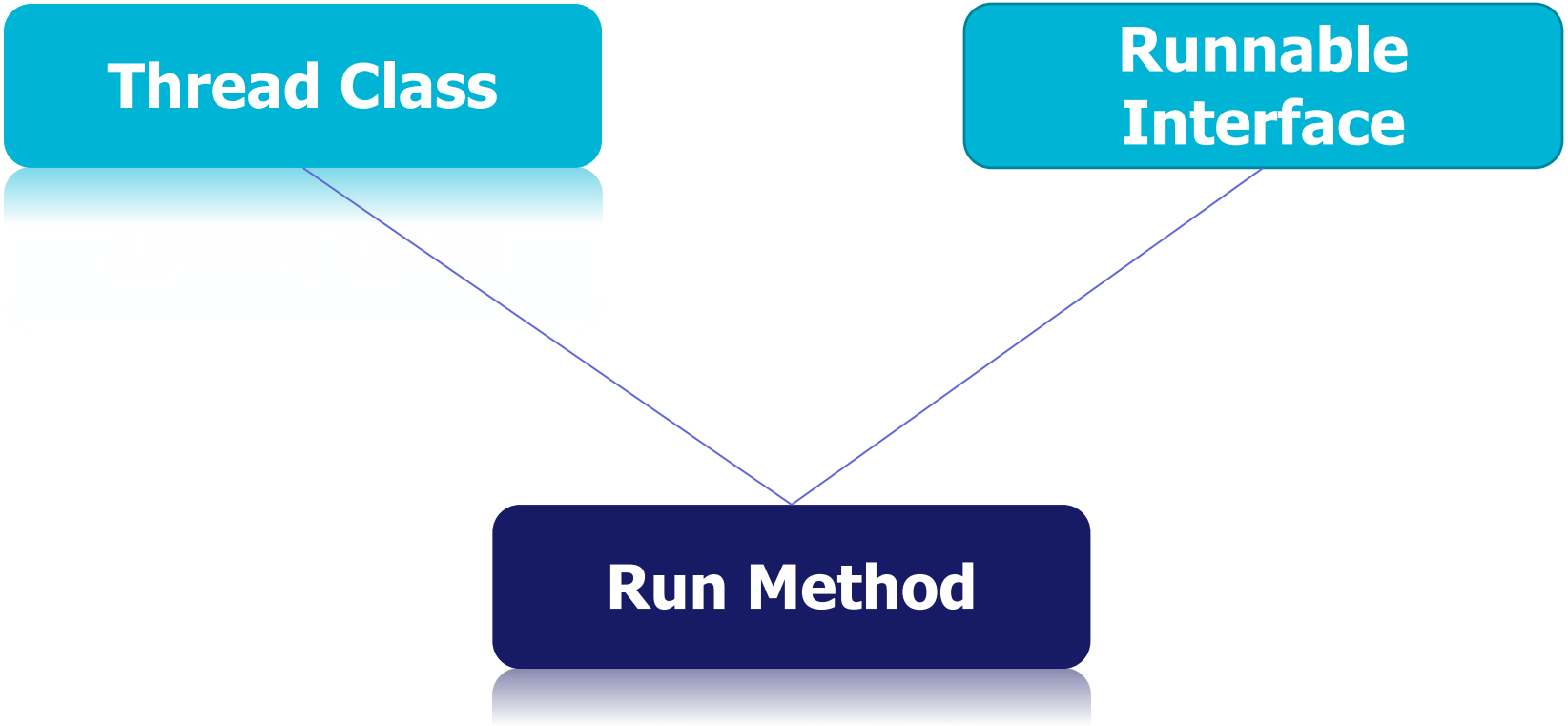
Ways to create own Thread

- implementing the ***Runnable interface***
- extending the ***Thread class***

Constructors

- Thread()
- Thread(Runnable target)
- Thread(Runnable target, String name)
- Thread(String name)

Java Threads



Threads in Java

- Java Virtual Machines support multithreading.
- Thread of execution in Java is an instance of class *Thread*. In order to write thread of excecution the class must inherit from this class and override the method run().

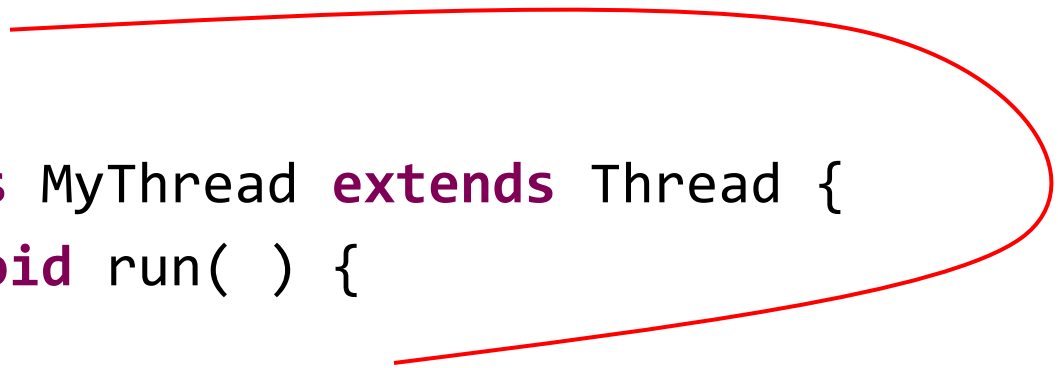
```
public class MyThread extends Thread {
    public void run( ) {
        // a long operation, calculation
        long sum = 0;
        for (int i = 0; i < 1000; i++) {
            sum += i;
        }
        System.out.println(sum);
    }
}
```

Threads in Java

To start a thread, you must create an instance of a derived class and call the inherited method start().

```
MyThread t = new MyThread( );
t.start( );

public class MyThread extends Thread {
    public void run( ) {
    // …
    }
}
```



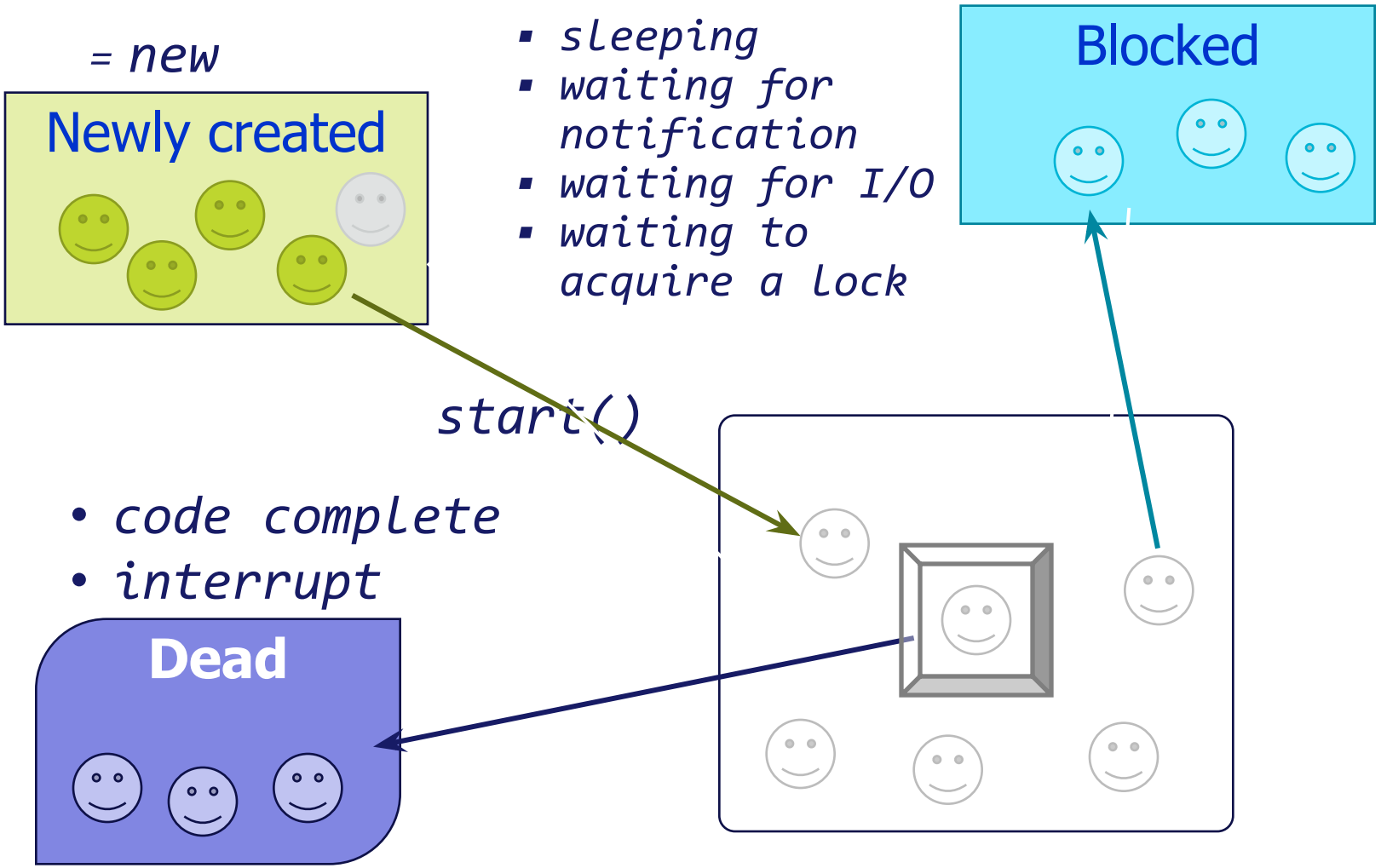
Threads in Java

- Since Java does not use multiple inheritance, the requirement to inherit from the Thread can lead to conflict.
- Sufficiently to implement an interface Runnable, which declared the method **void** run()

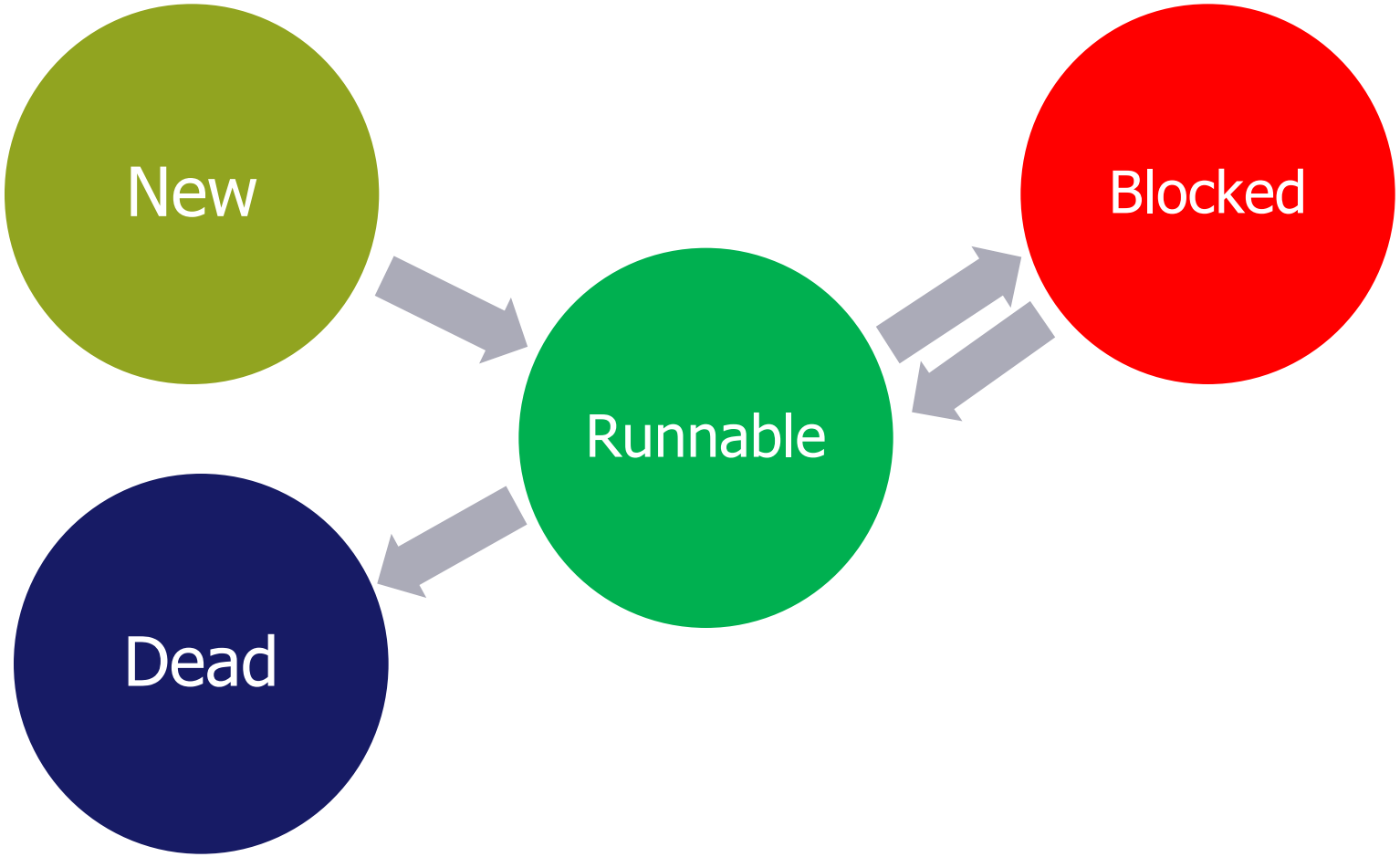
```
public class MyRunnable implements Runnable {
    public void run( ) {
        long sum = 0;
        for (int i = 0; i < 1000; i++) sum += i;
        System.out.println(sum);
    }
}
```

```
Runnable r = new MyRunnable( );
Thread t = new Thread(r);
t.start( );
```

Thread life cycle



State cycle



How to control threads ???

public static void yield();

- Method of java.lang.Thread
- Thread gives up CPU for other threads ready to run

public static void sleep (long millis) throws InterruptedException;

- Makes the currently running thread sleep (block) for a period of time
- The thread does not lose ownership of any monitors.
- InterruptedException - if another thread has interrupted the current thread.

public final void join();

- Wait until the thread is "not alive"
- Threads that have completed are "not alive" as are threads that have not yet been started

public final void setPriority (int newPriority);

- Set priority from 1 to 10
- New thread has default priority 5

Threads in Java

```
public class MyThread extends Thread {
   private int number;
   private int pause;

   public MyThread(int number, int pause) {
      this.number = number;
      this.pause = pause;
   }
   @Override
   public void run() {
      for (int i = 0; i < 5; i++) {
         try { sleep(pause);
         } catch (InterruptedException e) {}
         System.out.println("Thread " + number);
      }
}  }
```

Threads in Java

```
public class Example {
   public static void main(String[] args) throws
Exception {
      Thread t1 = new MyThread(1, 100);
      Thread t2 = new MyThread(2, 250);
      t1.start();
      t2.start();
      // t1.join();
      // t2.join();
      System.out.println("Thread main");
   }
}
```

Threads in Java

Also we can change the procedure to start the stream.

```
Thread t[ ] = new Thread[3];
```

```
for (int i = 0; i < t.length; i++) {
t[i] = new Thread(new MyRunnable( ), "Thread " + i);
// priority = 1, 4, 7
t[i].setPriority(Thread.MIN_PRIORITY
        + (Thread.MAX_PRIORITY - Thread.MIN_PRIORITY)
        / t.length * i);
t[i].start( );
}
```

```
Thread.MAX_PRIORITY = 10
Thread.MIN_PRIORITY = 1
Thread.NORM_PRIORITY = 5
```

Example

```
public class Run1 implements Runnable {
    @Override
    public void run() {
    for (int i = 0; i < 10000; i++) Appl.sum--;
    }
}
```

```
public class Run2 implements Runnable {
    @Override
    public void run() {
    for (int i = 0; i < 10000; i++) Appl.sum++;
    }
}
```

Example

no synchronization

```
public class Appl {
    public static int sum = 0;
    public static void main(String[ ] args) {
        Runnable r1 = new Run1( );
        Thread t1 = new Thread(r1);
        Runnable r2 = new Run2( );
        Thread t2 = new Thread(r2);
        t1.start( );
        t2.start( );
     Thread.yield( );
        System.out.println("Success, sum = " + sum);
    }
}
```

Synchronization in Java

Synchronized methods

Synchronized blocks

Methods

- wait
- notify
- notifyAll

Synchronized

- The keyword synchronized can be applied in two variants – to declare a synchronized-block and as a modifier of the method.
- If another thread has already installed a lock on object, the execution of the first stream is suspended. After this block it's executed.

```
public synchronized void myMethod() { … }
or
public void myMethod() {
    //some code
    synchronized(this) { //some code }
}
```

Synchronized

```
public class Run1 implements Runnable {
    @Override
    public void run() {
        for (int i = 0; i < 10000; i++) {
            synchronized(Appl.class) {
                Appl.sum--;
            }
        }
    }
}
```

Deadlock

When working with locks the possible appearance of deadlock should always be remembered – deadlock, which leads to stop responding the program.

```
public class DeadlockDemo {
public final static Object first = new Object();
public final static Object second = new Object();

public static void main(String s[]) {
   Thread t1 = new Thread() {
   public void run() {
      synchronized (first) {
         Thread.yield();
         synchronized (second) {
            System.out.println("Success!");
   }  }  }  };
```

Threads in Java

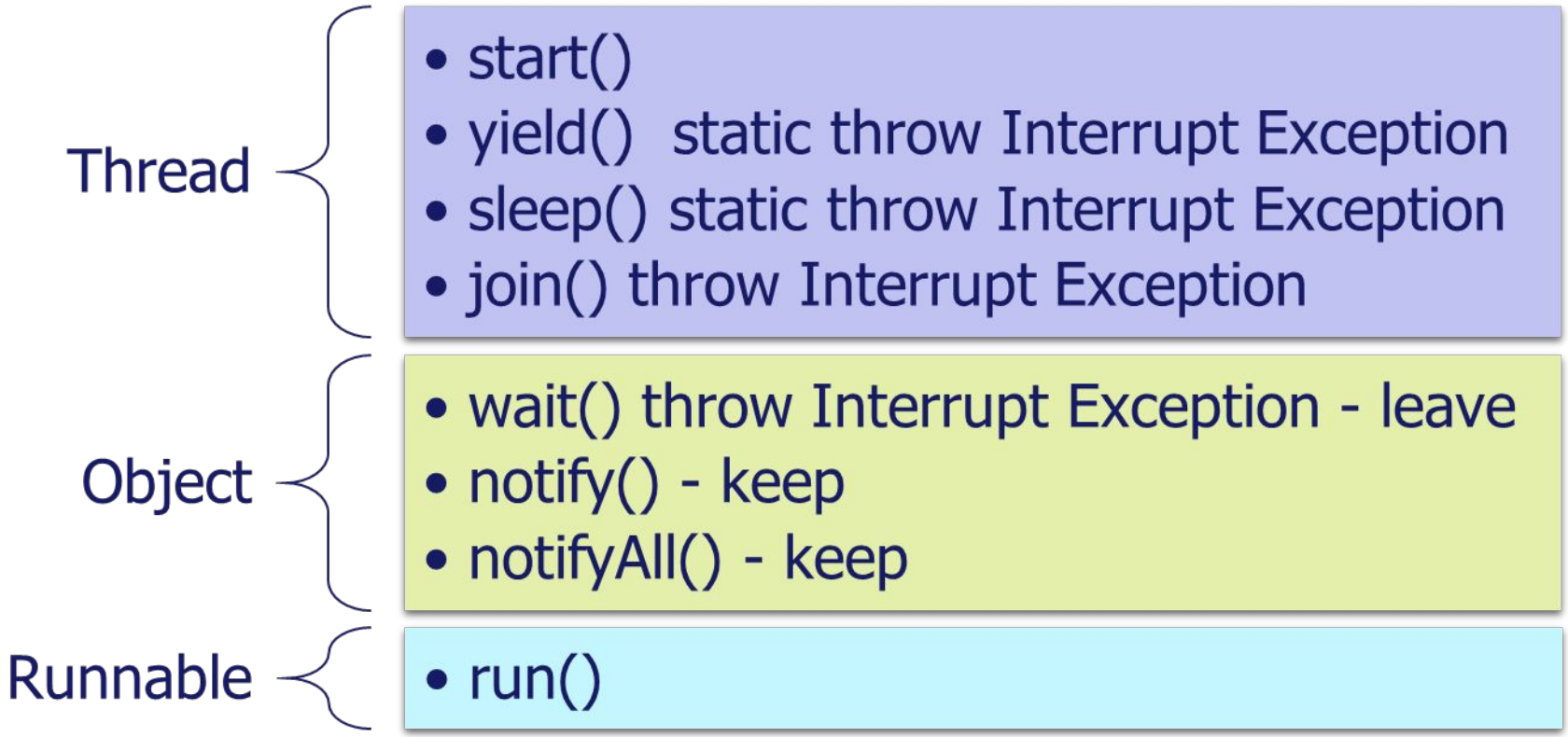
```
Thread t2 = new Thread() {
    public void run() {
        synchronized (second) {
            Thread.yield();
            synchronized (first) {
                System.out.println("Success!");
            }
        }
    }
};
t1.start();
t2.start();
}
}
```

wait() notify() notifyAll()

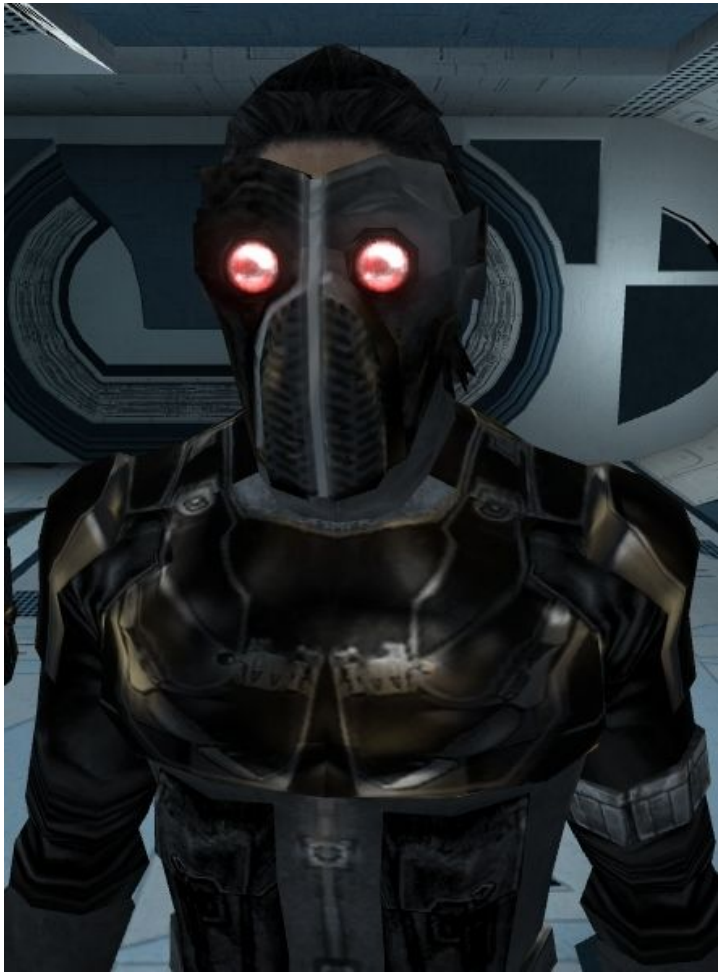
- Communication between threads
- Relative to an Object
- Example of using:

```
void todo() {
   synchronized(object){
      try{
         object.wait();
      } catch(InterruptedException e) {
         System.out.println("Interupted");
      }
   object.notify();
   object.notifyAll();
}
```

Thread summary



Daemon Threads



- Service Providers
- System kills all Daemon's when exits
- VM still "on the air" until last Thread dies
- Garbage Collector (finalize() call may never happen)

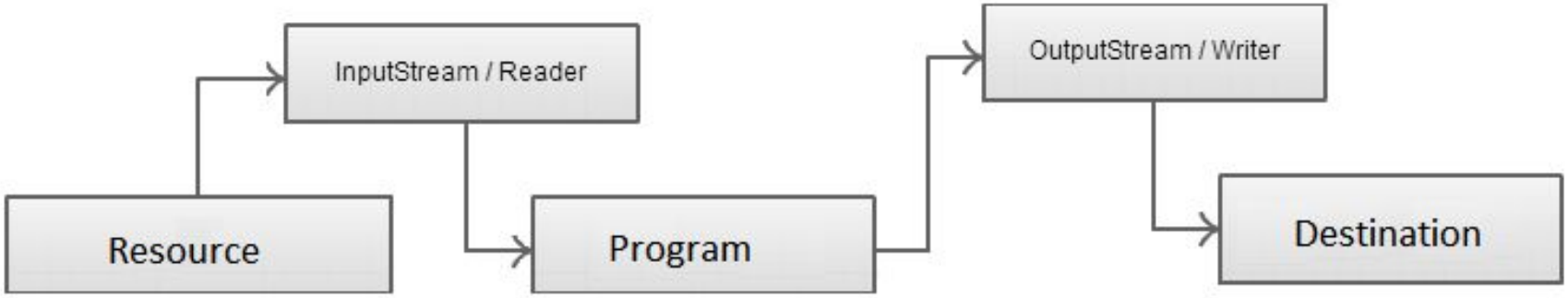
Typical threads work:

Goal

to block (***wait***) the **consumer** until the basket reaches some fruit

Data streams

- **IO API** (Input & Output) — Java API, designed for streaming.
- There are defined input and output streams in **java.io** (**InputStream** and **OutputStream**)
- Resource or Destination:
  - Console
  - File
  - Buffer etc.



Some classes of Java IO API

- InputStream / OutputStream
- Reader / Writer
- InputStreamReader / OutputStreamWriter

- FileInputStream / FileOutputStream
- FileReader / FileWriter

- BufferedInputStream / BufferedOutputStream
- BufferedReader / BufferedWriter

Some classes of Java IO API

- There are two abstract classes which base all the classes controlling by *the streams of bytes*:
  - **InputStream** (represents input streams)
  - **OutputStream** (represents output streams)
- To work with *the streams of characters* there are defined abstract classes:
  - **Reader** (for reading streams of characters)
  - **Writer** (for recording streams of symbols).
- There are a bridge from byte streams to character streams
  - **InputStreamReader** reads bytes and decodes them into characters using a specified charset
  - **OutputStreamWriter** writes characters to it are encoded into bytes using a specified charset

Java Input and Output Stream

```
public static void main(String[] args) {
  BufferedReader br = new BufferedReader(
      new InputStreamReader(System.in));
   int x = 0;
   System.out.print("Input number");
   try {
      x = Integer.parseInt(br.readLine());
   } catch (NumberFormatException | IOException e) {
      System.out.println("I/O Error.");
   }

   System.out.println("Number is " + x);
}
```

File Output

```
import java.io.*;
public class TestFile {
public static void main(String[] args) {
byte[] w = { 48, 49, 50 };
String fileName = "test.txt";
FileOutputStream outFile;
try {
outFile = new FileOutputStream(fileName);
System.out.println("Output file was opened.");
outFile.write(w);
System.out.println("Saved: " + w.length + " bytes.");
outFile.close();
System.out.println("Output stream was closed.");
} catch (IOException e) {
System.out.println("File Write Error: " + fileName);
}
} }
```

File Input

```
import java.io.*;
public class TestFileOutput {
public static void main(String[] args) {
byte[] r = new byte[10];
String fileName = "test.txt";
FileInputStream inFile;
try {
inFile = new FileInputStream(fileName);
System.out.println("Input file was opened.");
int bytesAv = inFile.available();     // Bytes count
System.out.println("Bytes count: " + bytesAv + " Bytes");
int count = inFile.read(r, 0, bytesAv);
System.out.println("Was readed: " + count + " bytes.");
System.out.println(r[0] + " " + r[1] + " " + r[2]);
inFile.close();
System.out.println("Input stream was closed.");
} catch (IOException e) {
System.out.println("File Read/Write Error: " + fileName);
} } }
```

File Input/Output

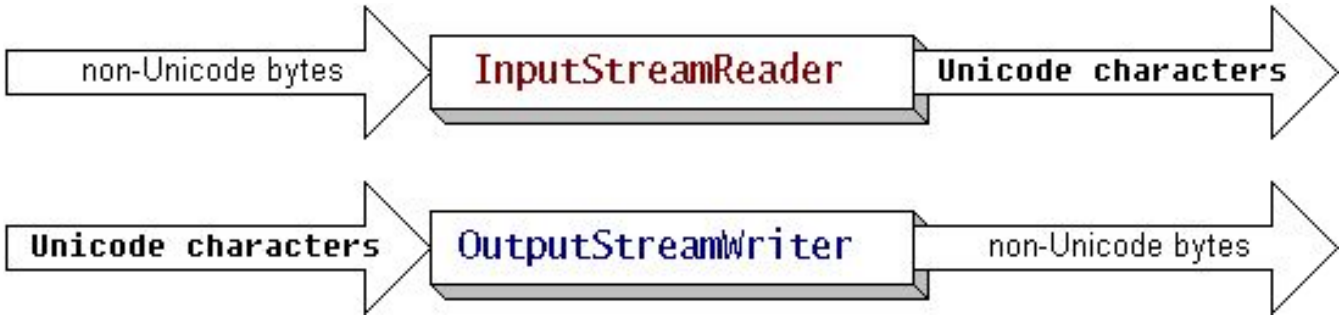
```
import java.io.*;
public class Test2 {
public static void main(String[] args) {
FileInputStream inFile1 = null;
FileInputStream inFile2 = null;
SequenceInputStream sequenceStream = null;
FileOutputStream outFile = null;
try {
inFile1 = new FileInputStream("file1.txt");
inFile2 = new FileInputStream("file2.txt");
sequenceStream =
    new SequenceInputStream(inFile1, inFile2);
```

File Input/Output

```
outFile = new FileOutputStream("file4.txt");
int readedByte = sequenceStream.read();
while (readedByte != -1) {
outFile.write(readedByte);
readedByte = sequenceStream.read();
}
} catch (IOException e) {
System.out.println("IOException: " + e.toString());
} finally {
try {
sequenceStream.close();
outFile.close();
} catch (IOException e) { }
}
}
}
```

File Input/Output

Reading from external devices – almost always necessary for buffer to be used
FileReader and FileWriter classes inherited from InputStreamReader and OutputStreamWriter.
The InputStreamReader class is intended to wrap an InputStream, thereby turning the byte based input stream into a character based Reader.



File Input/Output

```
public static void main(String[] args) {
String fileName = "file.txt";
FileWriter fw = null;
BufferedWriter bw = null;
FileReader fr = null;
BufferedReader br = null;
String data = "Some data to be written and readed\n";
try {
fw = new FileWriter(fileName);
bw = new BufferedWriter(fw);
System.out.println("Write data to file: " + fileName);
for (int i = (int) (Math.random() * 10); --i >= 0;) {
bw.write(data);
}
bw.close();
```

File Input/Output

```
fr = new FileReader(fileName);
br = new BufferedReader(fr);
String s = null;
int count = 0;
System.out.println("Read data from file: "
                    + fileName);
while ((s = br.readLine()) != null) {
System.out.println("row " + ++count
                + " read:" + s);
}
br.close();
} catch (Exception e) {
e.printStackTrace();
}
}
}
```

Practical tasks

1. Output text «I study Java» 10 times with the intervals of one second (Thread.*sleep(1000);*).
2. Output two messages «Hello, world» and «Peace in the peace» 5 times each with the intervals of 2 seconds, and the second - 3 seconds. After printing messages, print the text «My name is …»
3. Prepare mytext.txt file with a lot of text inside.

Read context from file into array of strings.
Each array item contains one line from file.
Complete next tasks:

1) count and write the number of symbols in every line.
2) find the longest and the shortest line.
3) find and write only that lines, which consist of word «var»

HomeWork

- Register at http://www.betterprogrammer.com/
- Install JDK 6 or configure your IDE to use it: http://www.oracle.com/technetwork/java/javase/downloads/java-archive-downloads-javase6-419409.html#jdk-6u45-oth-JPR
- Earn certificate with mark at least 75%

Homework

1. Run three threads and output there different messages for 5 times. The third thread supposed to start after finishing working of the two previous threads.
2. Cause a deadlock. Organize the expectations of ending a thread in *main*(), and make the end of the method main() in this thread.
3. Create a thread «one», which would start the thread «two», which has to output its number («Thread number two») 3 times and create thread «three», which would to output message «Thread number three» 5 times.
4. Create file1.txt file with a text about your career.

Read context from file into array of strings. Each array item contains one line from file.

Write in to the file2.txt

1) number of lines in file1.txt.
2) the longest line in file1.txt.
3) your name and birthday date.